PROGRAMMAZIONE 1 e LABORATORIO (A,B) - a.a. 2015/16
Prova scritta del 14/01/2016

SOLUZIONI PROPOSTE

Per ogni esercizio vengono proposte una o più soluzioni tra le tante possibili.

ESERCIZIO 1 (6 punti)
Dato l'alfabeto $\Lambda = \{a, b, c\}$ si definisca una grammatica libera che genera il seguente linguaggio:

$$\mathcal{L} = \left\{ a^k b^m c^n \ \mid \ k + n > m > 0 \right\}$$

Soluzione

```
S ::= AbC | aAb | bCc
A ::= aAb | aA | a
C ::= bCc | Cc | c
```

ESERCIZIO 2 (6 punti)
Si definisca in CAML, senza utilizzare ricorsione esplicita, una funzione

```
somma : int list -> int -> int
```

in modo che `(somma lis n)` restituisca:

- la somma degli elementi di `lis` che precedono la prima occorrenza di `n`, se `n` occorre in `lis`
- la somma di tutti gli elementi di `lis`, se `n` non occorre in `lis`

Ad esempio:

```
somma [2; 3; 4; 5; 4; 7] 4 = 5
somma [4; 2; 3; 4; 7] 4 = 0
somma [2; 3; 5] 4 = 10
```

Soluzione

```
let somma l n =
   let f x y = if x=n then 0 else x+y
   in
     foldr f 0 l;;
```

ESERCIZIO 3 (6 punti)
Si scriva in C una procedura che, prese due liste di interi, cancella dalla prima lista tutti gli elementi che compaiono anche nella seconda. Si suppongano predefiniti i tipi

```
struct el {int info; struct el *next;};

typedef struct el ElementoDiLista;
typedef ElementoDiLista *ListaDiInteri;
```

Soluzione
Definiamo dapprima la funzione ausiliaria `member` come segue

```
int member (int el, ListaDiInteri lis)
/* restituisce 1 se x occorre in lis, 0 altrimenti */
{
  int trovato = 0;
  while (lis != NULL && !trovato)
     if (lis->info == x) trovato = 1; else lis = lis-> next;
  return (trovato);
}
```

La procedura richiesta:

```
void canc (ListaDiInteri *l1, ListaDiInteri l2)
{
     ListaDiInteri prec=NULL, corr=*l;
     while (corr != NULL)
       {
         if (member(corr->info, l2))
            {ListaDiInteri aux = corr;
             if (prec==NULL)
                { *l1 = *l1 -> next;
                  corr= *l1;
                }
             else
                { corr = corr -> next;
                  prec -> next = corr;
                }
             free(aux);
            }
         else
            {
             prec = corr;
             corr = corr -> next;
            }
         }
}
```

ESERCIZIO 4 (6 punti)
Si scriva una funzione `C` che, dato un array a di dimensione dim, restituisce il seguente valore di verità:

$$\#\{j \mid j \in [0, dim) \ \wedge \ a[j] \leq 0\} > \#\{j \mid j \in [0, dim) \ \wedge \ a[j] > 0\}$$

Si ricorda che $\#A$ denota la cardinalità dell'insieme A.

Soluzione

```
int check (int a[], int dim)
{
   int contapos = 0;
   int j;
   for (j=0; j<dim;j++)
       if (a[j]>0) contapos = contapos + 1;
   return (dim > 2*contapos);
}
```

ESERCIZIO 5 (6 punti)
Dato il tipo degli alberi binari

```
type 'a btree = Void | Node of 'a * 'a btree * 'a btree
```

definire una funzione

```
firstneg: int btree -> int
```

che restituisce il primo valore (da sinistra verso destra) negativo incontrato in una foglia. Se l'albero non contiene valori negativi nelle foglie la funzione restituisce 0.

Soluzione
Definiamo una funzione ausiliaria che inserisce un elemento in un multiinsieme e la utilizziamo per inserire ricorsivamente gli elementi della lista nel multinsieme risultato.

```
let rec firstneg bt = match bt with
     Void -> 0 |
     Node(x, Void, Void) -> if x<0 then x else 0  |
     Node(x, lbt, rbt) when lbt<>Void or rbt<>Void
                          -> let v=firstneg lbt in
                                 if v=0 then firstneg rbt
                                        else v
```